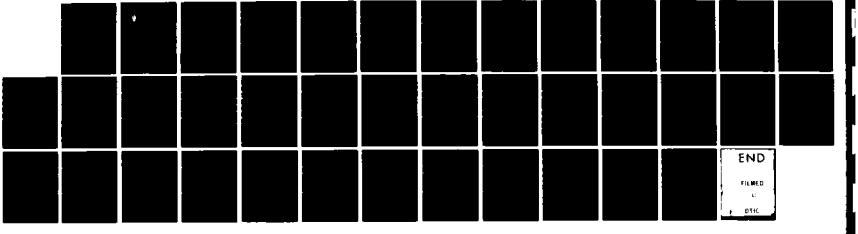
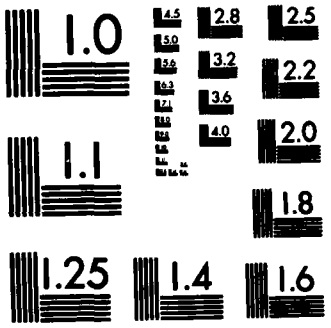
AD-A122 674 GEMINI MICROPROGRAMMER'S HANDBOOK(U) ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (ENGLAND) J KERSHAW SEP 82 RSRE-82015 DRIC-BR-85562 1/1

UNCLASSIFIED F/G 9/2 NL



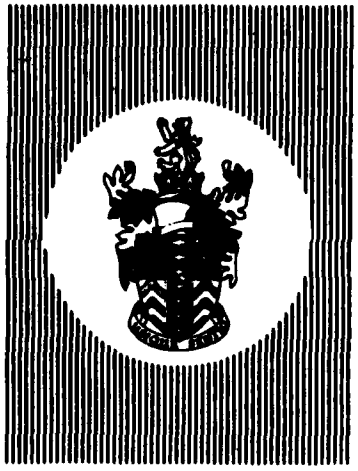


MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNLIMITED

BR85562

Report No. 82015



Report No. 82015

ROYAL SIGNALS AND RADAR ESTABLISHMENT,
MALVERN

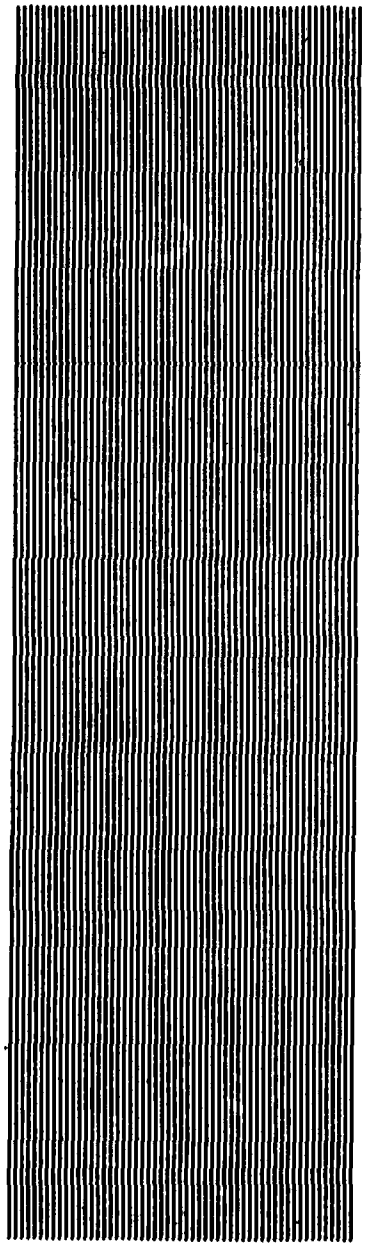
AD A122674

DTIC FILE COPY

GEMINI MICROPROGRAMMER'S HANDBOOK

Author: J Kershaw

DTIC
ELECTE
DEC 27 1982
S
E
D



PROCUREMENT EXECUTIVE, MINISTRY OF DEFENCE
RSRE
Malvern, Worcestershire.

82 12 20 164

September 1982

UNLIMITED

UNLIMITED

ROYAL SIGNALS AND RADAR ESTABLISHMENT

RSRE REPORT 82015

Title: GEMINI MICROPROGRAMMER'S HANDBOOK

Author: J Kershaw

Date: September 1982

SUMMARY

This Report is a compilation of documents relating to the GEMINI microprogrammed emulation system. Its purpose is to bring together as much as possible of the information needed by users of GEMINI, and particularly by microprogram writers. Information on the hardware and maintenance of GEMINI systems can be found in the GEMINI User's Handbook, published by Plessey Electronic Systems Research Ltd.

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A

DTIC COPY INSPECTED

Copyright
C
Controller HMSO London

1982

UNLIMITED

RSRE REPORT 82015

GEMINI MICROPROGRAMMER'S HANDBOOK

J Kershaw

CONTENTS

1 INTRODUCTION TO THE GEMINI EMULATION SYSTEM

2 GEMINI ASSEMBLY LANGUAGE

3 GEMINI MICROINSTRUCTION FORMATS

4 GEMINI DIAGNOSTIC PROGRAM UNIT

5 GEMINI SIMULATION PROGRAM

6 GEMINI UNPACKING UNIT

1. Introduction to the GEMINI emulation system.

GEMINI is a compact, relatively low cost, microprogrammed emulator. In its most basic form it occupies one 19 inch crate, including power supplies and up to 512k bytes of main memory. External memory may be connected, or the system can be interfaced to a DEC "Unibus" (TM) and use whatever memories or peripherals are connected to that bus. A memory suitable for direct connexion is available from Systems Reliability Limited. GEMINI itself has been developed jointly by RSRE and Plessey Electronic Systems Research, from an original RSRE design.

The most natural application for GEMINI is emulation of other computers: any target machine with up to 32 bits per word can be emulated, with average instruction times of 1 to 5 microseconds. The target can be either an existing computer (perhaps an obsolete machine with a large software investment) or a new architecture which does not yet exist in hardware. Multiple emulations are possible in the same system, allowing software proving for one target machine using the operating system and debugging tools of another.

As a computer in its own right, GEMINI is extremely fast but has limited direct addressing capability. Each of the two processors in a GEMINI system obeys a powerful multi-function instruction every 128nS, and can hold 4096 such instructions either in PROM or (with the optional Diagnostic Program Unit) in RAM. The two processors share access to a 768 word scratchpad memory, 32 bits wide, with a cycle time of 64nS. An additional 256 words of scratchpad addressing space are used to access peripherals: large buffered memory systems, a DEC "Unibus" interface, serial I/O controllers, or an "Unpacking Unit" which extracts variable bit fields from a 32 bit word in 64nS.

Applications of GEMINI outside emulation might include high speed communications, encryption/decryption, database management, and those areas of signal processing for which 32 bit fixed point arithmetic is adequate. Peripheral arithmetic units can be fitted if necessary. In many of these applications the two processors could be used with high efficiency.

A minimal GEMINI system would consist of a single processor, the scratchpad memory, and a peripheral interface. The microprogram would be held in PROM, mounted directly on the processor board. Adding a second processor increases the power of the system by a factor rather less than 2, at some cost in microprogram complexity, but provides a limited degree of fault tolerance: each processor can monitor the other's behaviour.

For microprogram development, one or two Diagnostic Program Units are needed. The small version of the DPU holds 256 microinstructions and is designed mainly for system testing; the large version holds the full 4096 instructions. Both use an Intel 8039 microprocessor to control loading and provide a "friendly" interface, and both allow the GEMINI processors to run at full speed. Data and commands to a DPU are sent via a standard V24 interface (up to 9600 baud) either from a conventional terminal or from a host computer.

GEMINI microprograms are written in a block-structured assembly language ("GEMIMA") which allows multiple source files and conditional assembly. Testing normally begins by running the microprogram on a simulated GEMINI, which resides in a conventional computer and provides detailed access to the internal state of the simulated machine. Both assembler and simulator are written in CORAL 66, and are available on the PDP11, TI990, and CTL Modular One.

For final testing the microprogram can be loaded into a DPU (or two DPU's in a twin processor system) and executed at full speed before any PROM's are committed. Loading can be either direct (e.g. from cassette tape) or via a host computer; in the second case the GEMINI Host Program (also in CORAL 66) is used to connect a host machine terminal to a number of DPU's and to take assembled microprograms from the host's filing system.

2. GEMIMA Assembly Language.

2.1 Introduction.

GEMIMA (as in Puddleduck) is a simple assembly language for the GEMINI microprogrammed emulator. Although GEMIMA programs have a superficial resemblance to intermediate level languages like PL/360, the similarity is only coincidental; GEMIMA is strictly a machine level language. The assembler itself runs on any computer accessible through the SDL "portable" CORAL compiler e.g. PDP10, TI9900, and occupies about 12kbytes upwards depending on object program size. It can take its input from any medium which allows repeated reading of the data, e.g. floppy disc or cassette tape, and it uses no work files. The output is a file suitable for input to the GEMINI writable microcode store or to the simulation program, but PROM programming data can also be produced.

Twin processor systems are programmed by producing two separate programs for the A and B processors, using common data declarations. These may be kept in a separate file, since the assembler allows a source program to occupy several files which may be on a variety of physical media.

2.2 Operating instructions.

On initiation the assembler requests the names of the source files in order, then the files to be used for output and diagnostics. The diagnostic file receives a list of variables and labels, with their addresses in hex and decimal, interspersed with any error messages. Labels are marked with an asterisk. The diagnostic output ends with a message indicating the number of stored constants generated, see Section 2.5. At the end of the run, the message RUN COMPLETE or RUN FAILED is output to the terminal.

All the operator's inputs are terminated by CR, and "backspace" (or "cursor left") and "cursor right" may be used to correct typing errors. The list of source files is terminated by a null file name, i.e. CR alone.

2.3 Basic principles of the language.

GEMIMA is a line-by-line language: every construct (including BEGIN and END, see below) requires a terminator. Carriage return or semicolon may be used interchangeably as terminators, except following a comment, and redundant terminators (e.g. blank lines) do no harm. Spaces (one or more) are allowed everywhere except within numbers or words, and all non-printing characters other than space and carriage return are ignored. Any string of characters beginning with % is ignored as a comment, up to (but not including) the next carriage return.

Integral numbers (up to 32 bits) may be written with any reasonable base, though the default is decimal:

```
1230000
8R777000177
16R00FF008A
2R11000001110
```

Numbers are right-justified and must be typed without spaces.

There are 16 reserved words: FINISH, HALT, A, X, Y, C, FROM, DECLARE, BEGIN, END, REPEAT, UNTIL, NULL, KEEP, STATUS, CONLIMIT. These may not be used in any context other than that defined in the syntax (Section 2.9). All other combinations of upper case letters and digits which begin with a letter are identifiers, invented by the programmer as variable names or instruction labels. Identifiers may be of any length, but the assembler remembers only the first 11 characters. Certain identifiers have special meanings (e.g. LEFT, AND, RETURN, CARRY) but these are distinguishable by context; their re-use as normal identifiers (while possibly confusing to the programmer) is not precluded.

FINISH is used to terminate the last file of a program, and HALT to terminate earlier files if any. A, X, Y, C stand for the four registers in the GEMINI processor: each has 32 bits, though the least significant 8 bits of C have a special use, see Section 2.7

FROM and DECLARE are used in variable declarations, see Section 2.4.

BEGIN and END are used to delimit blocks: any sequence of declarations and/or instructions may be enclosed between BEGIN and END, and any variables or labels declared within the resulting block will be inaccessible from outside. The scope rules are as in Algol or CORAL, i.e. forward reference is allowed to instruction labels but not to variables. The most local declaration of an identifier takes precedence over any other. Blocks may be nested to any depth.

A block is also the unit of conditional assembly. If the word BEGIN is followed by a list of one or more variable names, the following block and any blocks enclosed within it will be skipped (i.e. not included in the object program) if all the variables correspond to address 0 (see Section 2.4).

The following excerpt gives one of three different instruction sequences after L1 (neither block, first block, or both) according to the

declarations of TEST1 and TEST2, which must be in scope before BEGIN:

```
L1: BEGIN  TEST1, TEST2
     instruction
     instruction
    END
    BEGIN  TEST2
     instruction
    END
```

Blocks enclosed within a conditional block may themselves be conditional, but will always be skipped if the enclosing block is skipped. The contents of a conditional block which is skipped are not checked.

REPEAT is used to generate a block of identical instructions, usually jumps to an error routine:

```
REPEAT 100
```

will produce 100 repetitions of the preceding instruction. Any labels attached to the instruction are not repeated. This construction is of use mainly in writing test programs. An instruction can be repeated up to and including a defined address by writing after it (e.g.):

```
REPEAT UNTIL 4000
```

KEEP allows identifiers to be "promoted" to the surrounding block e.g.

```
BEGIN
DECLARE  P, Q, R
 instructions
KEEP  P, R
END
```

Labels or variables thus promoted appear in the surrounding block as if declared by the KEEP statement, and may be "kept" again if necessary.

STATUS is a built-in object which delivers up to 6 out of the first 16 condition signals, in a pattern determined by the wiring of a user supplied DIL plug. See Section 2.6 and the GEMINI User's Handbook.

CONLIMIT is used to change the position of the constant table in the shared data memory, see Section 2.5.

2.4 Labels and variables.

An identifier in GEMIMA may stand for an address in the instruction memory (a label) or an address in the shared data memory (a variable). All the identifiers declared within a given scope must be distinct. A label is

declared and associated with an address by prefacing it to an instruction:

```
LABEL1: instruction
```

Multiple labels are allowed. Notice that the label is associated with the next instruction written, so blank lines, variable declarations, or block headers may intervene without effect:

```
SUBROUTINE:
LABEL2: BEGIN  TEST1
        DECLARE WORK1, WORK2  % see below
        LABEL3: instruction
```

SUBROUTINE, LABEL2, and LABEL3 will correspond to the same instruction, even though LABEL3 has a different (narrower) scope.

Variables can be declared in several ways, all syntactically distinct from labels:

```
V1 = 0                  % address given explicitly -
                        % appropriate for peripheral
                        % addresses or for variables
                        % which select conditional blocks.

FROM 10: P,Q(2),R       % equivalent to P = 10; Q = 11;
                        % R = 13

V2 = Q                  % gives V2 = 11;  Q must be a
                        % variable which is in scope
                        % and already declared.

FROM P: I(8),J,K        % gives I = 10; J = 18; K = 19;
                        % restrictions on P as for Q.
```

The last form of declaration allows the assembler to allocate addresses itself, and is intended to be used by "library" subroutines which must create workspace independently of the program in which they are embedded:

```
DECLARE A1(5), WORK1, WORK2
```

Space is allocated sequentially, starting from location 0 of the data memory or from the end of the previous DECLARE construction. Other forms of declaration have no effect on the allocation, so that (assuming this is the first DECLARE construction):

```
DECLARE L, M, N(10)
FROM 2: F, G
DECLARE A2(6), A3(8)
```

will give L = 0, M = 1, N = 2, F = 2, G = 3, A2 = 12, A3 = 18.

2.5 Instructions.

GEMINI obeys instructions in the order written, unless the sequence

is broken by a Control Transfer Qualifier (see below). The first instruction obeyed when the system is switched on or reset will be the first written, at address 1 in the instruction memory. Instruction 0 is planted by the assembler and waits for the PWRFAIL condition (see Section 2.6) to become false. Each instruction has three main components:

condition operation part qualifiers

The first and last are optional; the operation part is always present and will be described first.

Every GEMINI instruction contains fields for a source register, an operation, a destination register, and a memory address. Thus the general form of an instruction which reads from memory is:

R1 := R2 OP VARIABLE

where "OP" is +, -, or one of the logical operators AND, OR, EQV. "R1" and "R2" are any of the four processor registers, A, X, Y, C. In practice the assembler can insert default settings for "R1 := " (to be the same as R2) or for "R2" (to be zero), so the following are legal:

```
A + VARIABLE          % for A := A + VARIABLE
X := -VARIABLE        % treated as 0 - VARIABLE
```

If R2 and the operator are omitted the operation is a simple register load:

```
C :=  VARIABLE(3)
C := +VARIABLE        % treated as 0 + VARIABLE
```

Notice the use of an integer offset, which can be signed - this is allowed with any variable, whether or not declared with a size specifier (e.g. DECLARE Q(2)), but not with labels. Its effect is simply to add or subtract (at assembly time) from the declared address of the variable.

With one minor exception (see "CARRY" below) the variable may always be read in complemented form:

```
A + *VARIABLE
X := -*VARIABLE(-14)
Y := C AND *VARIABLE
C := *VARIABLE        % etc.
```

In every case the memory contents remain unchanged. The logical operators AND, OR, EQV allow any combination of variable, source register, and result to be complemented:

```
 A := Y AND *VARIABLE
 C := *C EQV VARIABLE  % not equivalent, i.e. XOR
*X := A OR *VARIABLE   % X := complement of result
```

Note that "*Y AND VAR" is treated as "Y := *Y AND VAR" not "*Y := *Y AND VAR".

Whether or VARIABLE appears on the right hand side of an instruction, a constant is equally acceptable. The constant can extend to the full 32

bits, e.g.

```
X := A - 99
C := Y AND 16RFF800000
A EQV 2R1100100
```

The processor has a special mechanism for loading a 32 bit constant into a register, which the assembler will use whenever possible; when this is not possible the constant will be allocated a space in the shared data memory (from 767 downwards) and referred to as if it were a variable. The stored constants (if any) will be written once only, without duplication, when the program is initiated.

The position of the constant table in the data memory can be changed by use of the CONLIMIT directive, e.g.

```
CONLIMIT 700  or  CONLIMIT VARIABLE(99)
```

which specifies the highest address to be used. The default setting is 767. One (at least) of the programs for a twin processor system must use this mechanism to prevent the two constant tables overlapping in the shared memory. The CONLIMIT directive should appear once only, before any instructions.

A special form of constant gives access to the address of a label or variable:

```
Y := A + >LABEL    % forward reference to labels only
X := ↑VAR(3)       % offset with variables only
```

The characters > and ↑ are interchangeable.

Only one instruction changes the data memory contents. Its general form is

```
VARIABLE(7), R1  :=  R2
```

and it copies R2 to R1 and the variable. ", R1" may be omitted if not needed. In either case R2 remains unchanged. A memory location and a register may be set to zero by writing

```
VARIABLE, R1  :=  0
```

but in this case the register cannot be omitted. Complementing is not allowed in this operation.

In all these memory reference instructions, the variable name may be replaced by "@X" e.g.

```
 C  :=  @X        % no offset allowed!
 A  :=  Y + *@X
@X  :=  A
```

The effect of this is to use the present contents of the X register as the memory address; if the instruction also changes X the value used for addressing will be the value before the change.

The add and subtract operations (and the derived "load" operation) have variations illustrated by:

```
X  := C + VARIABLE + 1
Y - VARIABLE - 1          % "Y := " implied
A  :=  VARIABLE + 1
A  := -VARIABLE(-2) - 1
```

The variable may be complemented as usual. The final variation does not allow complementing:

```
Y  :=  VAR + CARRY
Y  :=  A + VAR + CARRY
Y  :=  A - VAR - CARRY
Y  :=  -VAR-CARRY
```

"CARRY" has value +1 if the CARRY condition is true when the instruction begins (see below) and 0 otherwise.

The remaining instructions operate on registers alone; although the memory address is still present it is ignored by the hardware and is not specified in the program. As before, the assignment may be omitted if the source and destination registers are the same.

```
C  :=  A LEFT 1      % left shift 1 place, fill with 0
X LEFT 1 + 1         % left shift and fill with 1
Y LEFT 1 + CARRY     % fill with CARRY condition
                     % (see below). Note that LEFT
                     % may also set CARRY.

A  :=  X + 1         % increment
Y - 1                % decrement
```

The source register may as usual be omitted when it will appear as zero. "A := LEFT 1 + CARRY" can be used to load the carry bit into a register.

```
A  :=  C             % copy. Neither may be omitted!
A  := *C             % copy complemented.  Ditto.

Y := STATUS          % load condition bits into a register
```

Finally, if no operation part is needed the word NULL can be used:

```
NULL, GOTO EXIT      % see Section 2.7
```

2.6 Conditions.

A condition may be typed at the beginning of any instruction. Its position emphasises the fact that the test is done before the operation part of the instruction, but it takes effect only at the end through a CALL, GOTO, or RETURN qualifier (see Section 2.7). If one of these qualifiers is written without a condition, it will be obeyed unconditionally. It is important to remember that the operation part of an

instruction, and any qualifiers other than CALL, GOTO, or RETURN, are always obeyed regardless of conditions. If the condition written is false, the instruction will be followed by the next one written.

There are 31 conditions (the 32nd is the default condition ?TRUE), 16 of which test the state of a register at the start of the instruction. They are separated from the operation part by a comma:

```
?A = 0,       ?A <> 0,       ?A >= 0,       ?A < 0,
?X = 0,       ?X <> 0,
?Y = 0,       ?Y <> 0,
?C = 0,       ?C <> 0,      --- bits 0..7 only

?AODD,        ?AEVEN,       --- bit 0 set or clear
?XODD,        ?XEVEN,

?ANORM,     --- "A" bit 31 different from "A" bit 30.
?ANAN,      --- "A" not approaching normal, i.e. bit 29
                is the same as bit 30.  Used in
                floating-point normalisation.
```

The remaining conditions test other aspects of the processor's status, or signals from outside:

```
?OVF        --- the last arithmetic operation which
                changed the A register overflowed. Almost
                any instruction involving +, -, or LEFT
                counts as arithmetic, including A := X+1,
                A := +VAR, A := -VAR, but not A := VAR or
                A := *VAR. "A := constant" does not count,
                even if the constant is signed, except in
                the special case where the constant has
                value +1 (however typed) which leaves
                both ?OVF and ?CARRY false.

?CARRY,     --- the last arithmetic operation which
                changed A caused carry. Operations other
                than arithmetic on A leave carry and
                overflow unchanged.

?CLOCK,     --- The system clock is about to be stopped by
                the diagnostic program unit (see Section 4).
                About 100 microseconds warning is given. No
                time-critical operation should be started
                while ?CLOCK is true.

?PWRFAIL,   --- the power supply is below nominal voltage.
                This condition becomes true about 1mS
                before power failure, and remains true
                for several mS after switch-on.

?DIAG,      --- signal from the optional diagnostic unit.
```

?EXT1..8, --- signals from peripheral interface units.

?WAIT2, --- the other processor has set WAIT2 (see Section 2.7). This condition can only usefully be tested in an instruction which contains a wait qualifier, since the signal is cleared when the wait ends.

?TIMEOUT, --- the waiting time limit (256 micro-instructions) has been exceeded, i.e. the other processor has failed. This condition remains set until the system is re-initialised.

Conditions may be spaced out to taste, as long as words are not broken. ? A = 0, or ? XEVEN, are acceptable but ? A ODD, is not.

2.7 Qualifiers.

These are subsidiary functions which can be performed at the same time as the operation part. Some modify the main operation, others are independent. They fall into groups which share the same bits in the instruction and may not be used together, but otherwise their number and order are unrestricted. Control transfer qualifiers are best written last, as they are chronologically the last to be executed. Commas are used as separators throughout.

Shift group:

, SR0 --- shift right, fill with zero. The source register of the main operation is shifted one place to the right before being operated on, filling bit 31 with 0.

, SRS --- as SR0, filling bit 31 with a copy of the original sign bit which is now bit 30.

, SRL --- as SR0, filling bit 31 with a copy of the initial value of the link flip-flop.

All shift qualifiers set the link flip-flop to the initial bit 0 of the source register, which would otherwise be lost. Note that the shift qualifiers operate after the register contents have been sent to memory,

so the instruction

VAR, A := X, SRS

will leave X unchanged, VAR = X, and A = X/2.

Count group:

, C-1 --- subtract 1 from the least significant 8 bits of C. The top 24 bits are left unchanged so 0 counts to +255 and +256 counts to +511. Note that "C := C-1" as an operation part would apply to all 32 bits. Conditions ?C=0 or ?C<>0 apply to the counting bits only; if used in the same instruction as the ,C-1 qualifier they refer to the initial value. If the operation part of the instruction specifies C as result register, the count qualifier is over-ridden and has no effect.

Wait 1 group:

, WAIT1 --- stop the processor until the other processor reaches an instruction containing either wait qualifier. The period of waiting is from zero (if the other is already waiting) to 256 whole instruction cycles, when the timeout operates.

Wait 2 group:

, WAIT2 --- as WAIT1. The processor can find out how it was released by using the ?WAIT2, and ?TIMEOUT, conditions.

Control transfer group. These qualifiers may be conditional: if the instruction in which they appear begins with a condition, the control transfer will be obeyed only if the condition was true at the moment the instruction began. If the condition was false, the succeeding instruction will be the *next* one written and the qualifier will have no effect - in particular, the subroutine stack will not be affected. If no condition is specified, any control transfer will be obeyed unconditionally.

, GOTO LABEL --- jump to the instruction with the corresponding label. Forward reference is allowed, but the label must be in scope (see Section 2.3).

, GOTO @Y --- jump to the instruction whose address was in the Y register at the start of the instruction.

, CALL LABEL --- as GOTO, but first push the address of the next instruction written on to a stack. The stack is in fact a 4 word memory addressed by a 2 bit up/down counter which will "wrap round" in either direction, so subroutine nesting needs care!

, CALL @Y --- as GOTO @Y, with return link.

, RETURN --- return from subroutine and pop stack.

2.8 Program Examples.

To transfer an array of constants into the data memory:

```
DECLARE TENPOWERS(5)     % table occupies 5 words

  X := >TENPOWERS        % data memory address
  Y := >DUMPER           % subroutine address

  A := 1, CALL @Y
  A := 10, CALL @Y
  A := 100, CALL @Y      % note 1 instruction -
  A := 1000, CALL @Y     % - for each constant
  A := 10000, CALL @Y

DUMPER: BEGIN            % subroutine
        @X := A
        X + 1, RETURN
        END
```

This technique is used by the assembler to set up the table of constants when a program is initiated.

The second example is a subroutine to multiply two 16 bit numbers together, giving a signed 32 bit product. The two operands are assumed to be in memory locations MULTIPLIER and MULTIPLICAND, occupying the least significant 16 bits of the GEMINI word with sign extension to 32 bits. The twos-complement result is assigned to the variable PRODUCT.

```
MULTIPLY:
  BEGIN
    DECLARE WORK1              % one workspace needed

      C := 0                   % sign flag for product

  % make both operands positive, keeping the sign of the
  % product as bit 0 of C

      A := MULTIPLICAND, CALL SIGNCHECK
      WORK1 := A
      A := MULTIPLIER, CALL SIGNCHECK

  % choose the smaller of the two operands as the multiplier

      A - WORK1
      ?A<0, A + WORK1, GOTO NOSWAP
      Y := A
      A := WORK1, GOTO SWAPDONE
    NOSWAP: Y := WORK1
    SWAPDONE: X := 0

  % multiplier is now in A, multiplicand in Y. The product
  % will accumulate in X as a positive 31 bit number

    LOOP: ?AODD, A := A, SR0, CALL ADD
      ?A<>0, Y := Y LEFT 1, GOTO LOOP

  % check the sign of the product

      A := C
      ?AEVEN, PRODUCT := X, RETURN   % positive result
      X := -PRODUCT
      PRODUCT := X, RETURN           % negative result

  % signcheck subroutine, returns modulus of A & records sign

    SIGNCHECK: ?A>=0, X := *A, RETURN
      A := X + 1, C - 1, RETURN

  % add subroutine, adds the shifted multiplicand to the
  % partial product in X

    ADD: WORK1 := Y                  % workspace needed
         X := X + WORK1, RETURN

  END
```

2.9 GEMIMA syntax.

Non-terminals are in lower case, all terminals in upper case including punctuation symbols such as COMMA, EQUALS. Empty alternatives are written <void>. Comments follow % as usual. Some occurences of ID correspond to built-in words such as LEFT, CARRY, AND, etc: these words are distinguishable by context and are not reserved.

```
program = statementlist FINISH;

statementlist = <void>,
                statement terminator statementlist;

terminator = NEWLINE,
             SEMICOLON;

statement = <void>,
             BEGIN skipcondition,
             END,
             KEEP keeplist,
             CONLIMIT INT,
             CONLIMIT ID offset,
             HALT,
             ID EQUALS INT,                    % declarations
             ID EQUALS ID offset,
             DECLARE declist,
             FROM INT COLON declist,
             FROM ID offset COLON declist,
             instruction,                      % all instructions
             ID COLON statement,               % anything labelled
             REPEAT INT,                       % "int" repetitions
             REPEAT UNTIL INT;                 % until address "int"

skipcondition = <void>,
                skiplist;

skiplist = ID offset,                          % note offset allowed
           ID offset COMMA skiplist;

keeplist = ID,
           ID COMMA keeplist;
```

```
declist = dec,
          dec COMMA declist;


dec = ID,
      ID OPEN INT CLOSE;                          % size in words


instruction = condition operation quals,
              condition COMMA operation quals, % comma is optional
              operation quals;


condition = QUERY ID,
            QUERY regname relop INT;              % A<0, X=0 etc.


reg = regname,
      ASTERISK regname;


regname = A,
          X,
          Y,
          C;


relop = EQUALS,
        LESS,
        GREATER EQUALS,
        LESS GREATER;                             % not equals


operation = var BECOMES reg,                      % write to memory
            var COMMA reg BECOMES reg,
            var COMMA reg BECOMES INT,            % INT is zero
            reg BECOMES reg,
            reg BECOMES reg optail,               % R1 := R2 OP VAR
            reg BECOMES loadtail,                 % R2 omitted
            reg optail,                           % R1 omitted
            NULL;


optail = PLUS object incpart,
         MINUS object decpart,
         logleft;                                 % LEFT and logic


logleft = ID object incpart;
```

```
object = var,
         const;

loadtail = const,
           PLUS const,
           MINUS const,
           var incpart,
           PLUS var incpart,
           MINUS var decpart,
           logleft,
           STATUS;


const = con,
        ASTERISK con;                          % complemented


con = INT,
      POINTER ID offset;


var = ID offset,
      AT X,
      ASTERISK ID offset,
      ASTERISK AT X;


offset = <void>
          OPEN sgint CLOSE;


sgint = INT,
        PLUS INT,
        MINUS INT;


incpart = <void>,
           PLUS INT,                           % INT must be 1
           PLUS ID;                            % ID is CARRY


decpart = <void>,
           MINUS INT,                          % as incpart
           MINUS ID;
```

```
quals = <void>,
        COMMA qual quals;

qual = ID,                          % single words
       ID ID,                       % CALL, GOTO ID
       ID AT Y,                     % CALL, GOTO @Y
       C MINUS INT;                 % C-1
```

3. GEMINI Microinstruction Formats.

All GEMINI microinstructions occupy 48 bits, in one of two formats:

Arithmetic and logic, in which the microinstruction is divided into 13 non-overlapping fields. The width and position of each field is fixed.

"Load constant", in which 6 of the fields are taken up by a 32 bit literal constant, one is ignored, and the remaining 6 are interpreted as for arithmetic and logic.

3.1 Arithmetic and logic format.

Each field is identified by a unique key character (see Sections 4 and 5) which gives a clue to its significance. In order from most to least significant bit, the fields are:

A (10 bits) Address of variable in shared data memory, or peripheral address. Held complemented.

X (1 bit) Index bit. If set, the A field is ignored and the initial contents of the X register (l.s. 10 bits) used instead.

F (5 bits) ALU function, see table below.

W (2 bits) WAIT bits. (WAIT2 is m.s.) If non-zero, the processor will wait till the other processor reaches a microinstruction with a non-zero W field.

> (2 bits) Right shift control. The shifter operates on the data sent from the source register to the ALU (but not on data sent to the shared memory) and if activated shifts one place to the right preserving the old bit 0 as LINK. For treatment of bit 31 see table.

D (2 bits) Destination register, see table.

S (4 bits) Source register, one bit only. See table.

? (5 bits) Condition. If the condition is true when the microinstruction begins execution the P, N, and J fields will be interpreted as described, otherwise these fields will be ignored and the microinstruction will be followed by the one at the next higher address.

- (1 bit) C-1 qualifier. If set, subtract 1 from the l.s. 8 bits of the C register. 0 will become 255, 256 will become 512. Note that the C=0 and C<>0 conditions apply to the l.s. 8 bits only.

C (1 bit) Constant. The microinstruction will be interpreted in "load constant" format, see below.

P (1 bit) Procedure: if the condition is true, push the address of the next microinstruction on to the stack.

N (1 bit) Next microinstruction address selector, see table.

J (12 bits) Jump address, used only if condition true and N = 3.

3.2 "Load Constant" format.

The A X F W > and J fields (in order from m.s. to l.s.) form a 32 bit literal constant, whose complement will be placed in the destination register. The S field is ignored, but the remaining fields are interpreted as normal. N = 3 should not be used, since J is part of the constant.

3.3 Numerical values of fields.

S = source register, D = destination register,
M = contents of data memory/peripheral location.

	F	> bit 31	?	S	D	N
0	D := STATUS	LINK	TRUE	zero	C	RETURN
1	NULL	old b31	OVF	C	A	@Y
2	NULL	zero	CARRY	A	Y	next
3	NULL	no shift	WAIT2	-	X	J field
4	D := S		TIMEOUT	Y		
5	D := S - 1		DIAG	-		
6	D := S - M		PWRFAIL	-		
7	D := S - M - 1		CLOCK	-		
8	D := S - M - CARRY		A<0	X		
9	D := S + M		A<>0	-		
10	D := S + M + 1		X<>0	-		
11	D := S + M + CARRY		Y<>0	-		
12	D := S LEFT 1		C=0	-		
13	D := S LEFT 1 + 1		AODD	-		
14	D := S LEFT 1 + CARRY		XODD	-		
15	D := S + 1		ANAN	-		
16	D := *S		A>=0			
17	*D := S AND M		A=0			
18	D := *S OR M		X=0			
19	D := -1		Y=0			
20	*D := S OR M		C<>0			
21	D := *M		AEVEN			
22	D := S EQV M		XEVEN			
23	D := S OR *M		ANORM			
24	D := *S AND M		EXT1			
25	*D := S EQV M		EXT2			
26	D := M		EXT3			
27	D := S OR M		EXT4			
28	D := 0		EXT5			
29	D := S AND *M		EXT6			
30	D := S AND M		EXT7			
31	M, D := S		EXT8			

4. GEMINI Diagnostic Program Unit.

A GEMINI system is controlled, not by a conventional "operator's panel", but by a simple command language input (either from a keyboard or from a host computer) to a Diagnostic Program Unit. The interface is to V24 standard, and will adapt automatically to any data rate in the following list: 110, 300, 600, 1200, 2400, 4800, 9600 baud. All outputs have even parity, but input parity is ignored.

The Diagnostic Program Unit exists in two forms, large and small. The large unit holds 4096 instructions which can be loaded, verified, inspected, and modified on-line. In addition it can manipulate the GEMINI system clock, and monitor the instruction addresses produced by the GEMINI processor. The small DPU holds 256 instructions, but contains in addition a 15 word data memory which can be addressed by the GEMINI processor as locations 1008 to 1022. The least significant 8 bits of location 1008 are continuously displayed on LED indicators, and the whole of this memory can be inspected by commands. The small unit also has the ability to monitor the GEMINI data bus, the data memory address, and the clock signals. It can load and execute on command (either from the keyboard or by pressing a button) a 256 instruction GEMINI program stored in an internal PROM, thus allowing the system to be checked without the use of a keyboard or host computer.

Commands to a DPU consist of a two letter mnemonic, followed in some cases by parameters, and terminated by carriage return. No action is taken until the terminator is input, and up to that point "backspace" (or "cursor left") and "cursor right" may be used to correct errors. Spaces are ignored everywhere except within decimal and hexadecimal parameters, see below. Illegal commands are reported with the message NO! but have no other effect. All inputs are echoed unless the cursor reaches either end of the line, where its movement will stop (but see below under LP and OP). A null command (CR alone) will provoke a new prompt.

When it is ready to receive a command, the DPU sends a prompt message to the operator. The last character of this message is always "BELL" (character 7); a host computer may use this as the trigger for the next command. The prompt message begins on a new line, and consists of a letter (G or D), followed by a decimal number, followed by one of the characters > + or -. The letter indicates whether the GEMINI program (in PROM) or the diagnostic program (in RAM) is being executed, the number is the address of the GEMINI instruction about to be obeyed, and the character shows whether the GEMINI clock is running, stepping under the control of the diagnostic system, or stopped.

When first switched on a DPU assumes state G 0>, allowing the GEMINI processor(s) to run on their internal PROMs, and prompts the operator at

110 baud. The input can be synchronised to the data rate in use by sending a succession of "delete" characters (3 per step in the table of data rates is sufficient) followed by CR, which should provoke NO! and a legible prompt. If the DPU occupies the right-hand slot in the crate, the prompt will be indented 10 columns (or 40 if option 1 is set, see below).

A GEMINI system can operate without a DPU, provided its program is in PROM. In this case the system starts automatically at instruction 0 when powered on.

4.1 Commands and parameters for the small DPU.

<integer> is an unsigned decimal number, terminated by any non-digit e.g. space or CR. Leading zeros may be omitted.

<hexnumber> is a hexadecimal number of up to 8 digits, terminated by any character other than 0..9 or A..F. Leading zeros may be omitted.

<boolean> is a single letter T or F.

Unless otherwise stated, commands may be used at any time and leave the system state unchanged.

LP <integer>

Load program using data rate <integer> baud from the table above. The parameter may be omitted if the data rate is the same as for commands, and the rate will revert automatically at the end of the program input. Input data is as produced by the GEMIMA assembler, and may be echoed (with normal error-correcting facilities) or not according to the setting of option 2 (see under OP below). After completion the system is left stopped and reset, with the diagnostic program (i.e. the one just loaded) selected (state D 0-).

Error messages:

NO!
: Data rate not acceptable.

TOO BIG
: Program is too big for memory, 256 or 4096 instructions.

INPUT ERROR
: Input checksum has failed.

02 1A FAILED
: A memory location cannot be read back correctly, in this case at byte 2 bits 1, 3, and 4. Only the last error found is reported.

FAILED
: Program sumcheck fails - see VP.

RS

Reset the GEMINI instruction counter to zero, and continue in the same state as before.

GO

Allow system clock to run freely. State becomes G n> or D n>; the instruction address in this and subsequent prompts will be valid but (obviously) out of date when printed. If several prompts give the same or related addresses while the clock is running, the microprogram is probably looping.

ST

Stop system clock. State becomes G n- or D n-. This command leaves the GEMINI processor in the second quarter of its instruction cycle (as does RS if the system is already stopped), when all the system buses carry significant data. To do this it may have to step the system clock up to 3 phases. If the clock fails to step on correctly the message CLOCKS is printed; this message may also appear while stepping (see SN and SU).

RI <integer>

Read the instruction at address <integer> (in either GEMINI or diagnostic memory, as currently selected), decompose it into 14 separate fields, and print it with key characters e.g.

A3 X0 F9 W0 >3 D1 S2 ?0 -0 C0 P0 N3 J366 #00F6 CE91

The first 13 fields are in decimal, and correspond to the microinstruction fields described in Section 3:

A	10 bits	Data memory/peripheral address (complemented)
X	1 bit	Use contents of X as data address
F	5 bits	ALU function - see table in Section 3
W	2 bits	WAIT2 (m.s.) and WAIT1 (l.s.)
>	2 bits	Right shift control
D	2 bits	Destination register
S	4 bits	Source register (one bit only set)
?	5 bits	Condition
-	1 bit	C-1 qualifier
C	1 bit	Constant - load 32 bit number to register
P	1 bit	Procedure (or Push) i.e. CALL qualifier
N	2 bits	Next microinstruction address selector
J	12 bits	Jump address

The last field is in hexadecimal, and is the complement of the 32 "constant" bits in the microinstruction. It is significant only if the C field is 1.

LI <integer> <field list>

Change the specified fields of the instruction at address <integer>. Each field is typed as a key character (see RI) followed by a <hexnumber> (# only, note no spaces) or an <integer> (all others). Fields may appear in any order, and fields which do not appear in the list are left unchanged. Spaces before and after the key character are optional, unless a <hexnumber> is followed by a C or D field when a space is essential e.g. #1FFF C1 D2. An A or # field will be

complemented before being stored. The # field overlaps the AXFW>J fields; in cases of conflict the last to be typed "wins". See example in Section 5.

NI <integer>
Print the instruction at address <integer> as for RI, then replace it with the null instruction A := A. The object is to reduce the number of fields which must be typed in a subsequent LI command. NI or LI will give the error message NO! if the GEMINI program (i.e. PROM) is selected.

DP
Select diagnostic program (i.e. RAM). Changes state to D 0-.

GP
Select GEMINI program (i.e. PROM). Changes state to G 0-.

RF
Read flags. Prints external conditions as 1F, 2T, 3F etc. An external condition may become TRUE as a result of an LF command or because a peripheral has set it - logical OR applies. Flag 9 is DIAG.

LF <integer> <boolean>
Load flag <integer> with <boolean>. The Diagnostic Unit sets all flags initially FALSE, but flags 1 to 8 may still be set by peripherals.

CL
Print clock states, in the order T1, $\overline{T1}$, T3, $\overline{T3}$, WAIT1, WAIT2, INITIALISE, e.g. 0101 01 1. Note that the last three are false if high. The clock state shown (0101) is the one which follows use of the ST command.

OP <integer>
Set options. <integer> is a sum of values as follows:
1: If set, cursor operations appropriate to a VDU terminal. Otherwise, as for a hard-copy terminal.
2: If set, "LP" data will be echoed with normal cursor movement facilities. Otherwise no echo, data will be accepted as fast as it can be sent at the selected data rate.
4: If set, "LP" size limit 4096 instructions. Otherwise 256.
8: If set, LED sampling of location 1008 disabled. This prevents any manipulation of the GEMINI system clock unless explicitly commanded. If two Diagnostic Units are used in a twin processor system, one must have option 8 set.
The initial setting of the options is controlled by 4 switches labelled 1, 2, 3, 4 for options 1, 2, 4, 8.

RD <integer>

Read a location of the 15 word data memory as 8 hex digits. <integer> is masked to 4 bits and used as the address; location 15 exists but usually contains rubbish. This memory can be written only by the GEMINI processor adjacent to "this" Diagnostic Unit - if the other processor is present it may have a Diagnostic Unit of its own but will not "see" the data memory in this one.

VP

Verify program. The program in the GEMINI or diagnostic memory (whichever is selected) is sumchecked and compared with the sumcheck from the last LP operation. Error message: FAILED.

PT

Push-button test. Load the diagnostic program memory from the internal PROM, reset, and enter state D 0>. Message PB TEST is output, and FAILED if the program does not sumcheck after loading. Result of pressing button is identical.

The remaining 6 commands are accepted only if the GEMINI clock is either stopped or stepping under diagnostic control. Otherwise the message RUNNING is given.

SC

Step clock one data-memory cycle, i.e. 2 clock phases, half a processor cycle. This allows the other processor in a twin processor system to be monitored to a limited extent, using the RM and RB commands.

RM

Read the GEMINI data memory address in decimal, with READ or WRITE as appropriate. The address is generated by the adjacent processor only in the first half of its instruction cycle, i.e. in clock states 0110 or 0101. In the second half of the cycle the address comes from the other processor (if it is present).

RB

Read the GEMINI data bus as 8 hex digits. In the clock state following ST (0101) the data bus carries either the output from the processor (WRITE) or the response from the memory/peripheral (READ). The other processor may be monitored by preceding the RB or RM command with SC. Note that ST will restore the clock state to 0101 even if the system is already stopped.

SN <integer>
Step on <integer> instructions, at an effective instruction time of about 130 microseconds (the speed may improve in later versions). This command may be abbreviated to <integer> alone, and SN0 (or just 0) is treated as SN1. The largest integer acceptable is 65535.

SU <integer>
Step until the instruction at address <integer> is about to be obeyed. SN and SU set the state to D n+ or G n+, and can be aborted while in progress with an ST command. When either command is completed the state reverts to D n- or G n- and a new prompt appears.

4.2 Limitations of the large Diagnostic Program Unit.

The large Diagnostic Unit lacks the hardware to carry out the RF, LF, CL, RD, PT, RM, and RB commands. It will accept the commands, since it uses the same software, but will give meaningless results. PT will overwrite the program memory with rubbish and then attempt to execute it. ST will leave the clock state random.

5. GEMINI Simulation Program.

The simulator is a CORAL program which takes a GEMIMA assembler output and "runs" it interpretively under operator control. The apparent instruction time is about 660 microseconds, or one fifth the speed of the DPU in stepping mode. Its user interface is as similar as practicable to the GEMINI Diagnostic Program Unit interface. Some of the commands to the DPU have no equivalents in the simulator (e.g. those which manipulate the internal GEMINI clock, and those whose main purpose is to help find hardware faults); conversely the simulator has some facilites which are not available in the DPU because of hardware limitations. There is one fundamental difference between the two which is not obvious from a simple list of facilities: the Diagnostic Program Unit will accept user inputs at any time, whereas the Simulator "goes dead" while it is obeying a command.

The rules for typing inputs to the Simulator are the same as for the DPU, except that on some host systems the cursor behaviour at the ends of a line may differ. The prompt message is rather different: the letter is omitted (since only one instruction memory is present) and the state character is always "-". The final character, as always, is "BELL".

Errors during execution (e.g. stack full or empty, program address outside the range loaded) cause a message to be printed and the user prompt to reappear. The instruction address will be that of the offending instruction.

The simulator has two peripherals built in: a standard unpacking unit occupying locations 768 to 799, and an external interface (base address 832, adjustable by macro at compile time) connected to a simulated Systems Reliability memory of (currently) 1024 bytes including tags. All the tag and byte handling mechanisms work except that none of the external conditions will be set - the memory's response appears to be instantaneous.

Peripheral addresses outside the range of the built-in pair generate a message to the operator, either

<instr address> - READ <peri address> INPUT =

or <instr address> - WRITE <peri address> DATA <hexnumber>

In the first case the operator must input a hex number. After either message the Simulator will prompt for a new command.

Error messages.

NO!
Command is illegal, e.g. mnemonic not recognised, not enough parameters, address specified is out of range, instruction at address specified (command RI only) has not been written.

TOO MANY RETURNS!
The subroutine stack has been exhausted. The stack is actually a 4 word RAM addressed by a 2 bit counter, as in the AMD 2909 hardware; CALLing more than 4 deep is allowed as long as the over-written return addresses are no longer needed. The simulator marks each return address as it is used and gives this message if any are used twice.

MORE THAN 1 SOURCE REGISTER!
Source register field is not a single bit. Usually indicates a jump outside the loaded program, since unwritten instructions are marked with source register = 15.

TOO BIG!
Program being loaded will not fit in the simulator store (1024 instructions in the Modular One version).

SUMCHECK FAILED!
On program loading. The program can be run, at your own risk.

5000 INSTRUCTIONS!
Limit for SU command. Input another SU to continue.

EXTERNAL ADDRESS TOO BIG!
An external memory cycle has been started with an address outside the Simulator range (1024 bytes in the Modular One version).

5.1 Differences between Simulator and DPU commands.

The Simulator does not recognise the commands GO, ST, DP, GP, CL, OP, VP, PT, SC, RM, RB.

The commands LP, RF, LF, RD, SU have differences of interpretation described below:

LP <filename>
The parameter is a file name in whatever convention the host system uses, specifying the source of the program. The controlling terminal may be used, though the GEMIMA output format is not designed for manual input. Possible error messages are TOO BIG! and SUMCHECK FAILED!

RF
14 booleans are printed: external conditions 1 to 8, DIAG, TIMEOUT, POWERFAIL, CARRY, OVERFLOW, LINK.

LF <integer> <boolean>
Any of the 14 flags may be loaded, but CARRY, OVERFLOW, and LINK are likely to be modified by the simulated program.

RD <integer>
Any address from 0 to 1023 can be read, though peripherals may not give meaningful results.

SU <integer>
If neither the address specified nor a peripheral operation has been reached after 5000 instructions have been processed, the Simulator will re-prompt.

The remaining commands are peculiar to the Simulator:

LA <hexnumber>
Load GEMINI "A" register. Similarly LX, LY, LC.

RA
Print GEMINI "A" register. Similarly RX, RY, RC.

CA <integer> <hexnumber>
Load the instruction "A := <hexnumber>" into the program memory at address <integer>. Similarly CX, CY, CC.

LD <integer> <hexnumber>
Load the data memory with <hexnumber> at address <integer>. Address range 0 to 767 only.

LE <integer> <hexnumber>
Load external memory byte at address <integer> with <hexnumber>. Only 9 bits are significant, bit 8 is the tag.

RE <integer>
Print an external memory byte (including tag) as 3 hex digits.

LT <filename>

Load target machine memory, i.e. do a succession of LE commands using data from the file. The format consists of a list of <integer> <hexnumber> pairs, each pair terminated by either carriage return or semicolon. Spaces may appear anywhere except within the integer. The terminator is an asterisk appearing in place of the next integer.

```
e.g.  0  21;  1  02
      2  0B;  3  12
      4  02;  5  C3
      *
```

The error message NO! appears if the format is incorrect or an address is out of range.

TE

Terminate the simulation program.

5.2 Example of a simulation run.

```
GEMINI SIMULATOR
 0-LP JK2/MULTIPLY
 0-RI3
      A1008 X0 F31 W0 >3 D1 S2 ?0 -0 C0 P0 N3 J0 #FC20 CFFF
 0-RI 99
NO!
 0-NI99
      A1023 X0 F0 W0 >0 D0 S0 ?0 -0 C0 P0 N0 J0 #FFFF FFFF
 0-LI99 A300 F19 S8 -1 N3 ?23 J106
 0-RI99
      A300 X0 F19 W0 >3 D1 S8 ?23 -1 C0 P0 N3 J106 #4B26 CF95
 0-LF2 T
 0-RF
1F  2T  3F  4F  5F  6F  7F  8F  9F 10F 11F 12F 13F 14F
 0-LX 01234
 0-LY3
 0-RX
0000 1234
 0-LD0 1111AAAA
 0-30
 3-WRITE 1008 DATA 0000 369C
 0-RD0
0000 0003
 0-TE
****
```

6. GEMINI Unpacking Unit.

The Unpacking Unit (Field Extractor, Barrel Shifter) is a peripheral which allows GEMINI microprograms to extract bit fields (from 1 to 32 bits wide) from either of two source registers in a single memory cycle.

It recognises any of a block of 32 addresses, starting at either 768 or 800 (switch-selectable). Fifteen distinct bit fields are available at any one time, but the fields may be changed dynamically by writing new values into the shift and/or mask tables (see below). Both tables are undefined at power-on.

WRITE to relative address:

0	Load source register 1.
1..15	Load shift table entry for field 1..15. The shift is a 5 bit number specifying the least significant bit of the field required.
16	Load source register 2.
17..31	Load mask table entry for field 1..15. The mask is a 32 bit pattern whose complement will be ANDed with the shifter output, thus the mask should be right-justified and contain 0's for the wanted bits. Wanted bits need not be contiguous but should not extend beyond source register bit 31! Notice that GEMIMA allows almost any constant or variable to be complemented, see example.

READ from relative address:

0	Undefined. A field spanning all 32 bits can be used to read the whole of a source register if necessary.
1..15	Read field 1..15 from source register 1. The pattern in the source register is shifted right and the result ANDed with the complemented mask. The final result is complemented again and delivered to the processor. The source register itself is left unchanged.
16	Undefined.
17..31	Read field 1..15 from source register 2.

Example - use of unpacking unit.

```
FROM 768:  SOURCE1, SHIFT1
FROM 784:  SOURCE2, MASK1

     S1FIELD1 = SHIFT1
     S2FIELD1 = MASK1


     A := 17              % bits 17 to ...
     SHIFT1 := A
     A := *16R3FF         % ... 26 (10 bits)
     MASK1 := A           % note mask is complemented

  ----------------

     Y := ANYTHING
     SOURCE1 := Y
     A := X + *S1FIELD1   % add bits 17..26 of "ANYTHING"
                          % note field also complemented
```

DOCUMENT CONTROL SHEET

Overall security classification of sheet ..UNCLASSIFIED.....UNLIMITED.....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference	3. Agency Reference	4. Report Security Classification
	REPORT 82015		Unclassified
5. Originator's Code (if known)	6. Originator (Corporate Author) Name and Location		
	ROYAL SIGNALS AND RADAR ESTABLISHMENT		
5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location		

7. Title

GEMINI MICROPROGRAMMERS HANDBOOK

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials	9(a) Author 2	9(b) Authors 3,4...	10. Date	pp. ref.
KERSHAW J				

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution statement

UNLIMITED

Descriptors (or keywords)

GEMINI

continue on separate piece of paper

Abstract

This Report is a compilation of documents relating to the GEMINI micro-programmed emulation system. Its purpose is to bring together as much as possible of the information needed by users of GEMINI, and particularly by micro-program writers. Information on the hardware and maintenance of GEMINI systems can be found in the GEMINI User's Handbook, published by Plessey Electronic System Research Ltd.

END

FILMED

2-83

DTIC